Interpreting Black Box Predictions using Fisher Kernels

Rajiv Khanna
UC Berkeley

Been Kim
Google Brain

Joydeep Ghosh
UT Austin

Oluwasanmi Koyejo
UIUC

Abstract

Research in both machine learning and psychology suggests that salient examples can help humans to interpret learning models. To this end, we take a novel look at black box interpretation of test predictions in terms of training examples. Our goal is to ask "which training examples are most responsible for a given set of predictions"? To answer this question, we make use of Fisher kernels as the defining feature embedding of each data point, combined with Sequential Bayesian Quadrature (SBQ) for efficient selection of examples. In contrast to prior work, our method is able to seamlessly handle any sized subset of test predictions in a principled way. We theoretically analyze our approach, providing novel convergence bounds for SBQ over discrete candidate atoms. Our approach recovers the application of influence functions for interpretability as a special case yielding novel insights from this connection. We also present applications of the proposed approach to three use cases: cleaning training data, fixing mislabeled examples and data summarization.

1 Introduction

It has long been established that using examples to enable interpretability is one of the most effective approaches for human learning and understanding [21, 4, 15]. The ability to interpret using examples from the data can lead to more informed decision based systems and a better understanding of the inner workings of the model [17, 16]. In this work, we are interested in finding data points or prototypes that are "most responsible" for the underlying model making specific predictions of interest. To this end, we develop a novel method that is model agnostic and only requires an access to the function and gradient oracles.

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

In a more formal sense, we aim to approximate the empirical test data distribution using samples from the training data. Our approach is to first embed all the points in the space induced by the Fisher kernels [13]. This provides a principled way to quantify closeness of two points with respect to the similarity induced by the trained model. If two points in this space are close, then intuitively the model treats them similarly. We formally show that influence function based approach to interpretability [17] is essentially doing the same thing.

Thus, our goal is to find a subset of the training data such that, when also embedded in a model-induced space, is *close* to the test set in the distribution sense. We build this subset from the training data sequentially using a greedy method called Sequential Bayesian Quadrature (SBQ) [22]. SBQ is an importance-sampling based algorithm to estimate the expected value of a function under a distribution using discrete sample points drawn from it. To the best of our knowledge SBQ has not been used in conjunction with Fisher kernels for interpretability. Moreover, we leverage recent research in discrete optimization to provide novel convergence rates for the algorithm over discrete atomic sets. Our analysis also yields novel and more scalable algorithm variants of SBQ with corresponding constant factor guarantees.

Our key contributions are as follows:

- We propose a novel method to select salient training data points that explain test set predictions for black box models.
- To solve the resulting combinatorial problem, we develop new faster convergence guarantees for greedy Sequential Bayesian Quadrature on discrete candidate sets. One novel insight that results is the applicability of more scalable algorithm variants for SBQ with provable bounds. These theoretical insights may be of independent interest.
- We recover the influence function based approach of Koh & Liang [17] as a special case. This connection again yields several novel insights about using influence functions for model interpretation and training side adversarial attacks. Most importantly, we establish the importance of the Fisher space for robust learning that can hopefully lead to promising future research directions.

- To highlight the practical impact of the our interpretability framework, we present its application to three different real world use-cases.

Related work: There has been a lot of interest lately in model interpretation in various ways and their corresponding applications. Thus, we focus our related work on the subset of most closely related research. Our approach has a similar motivation as Koh & Liang [17], who proposed the use of influence functions for finding the most *influential* training data point for a test data point prediction. The intuition revolves around infinitesimally perturbing the training data point and evaluating the corresponding impact on the test point. The method is only designed for single data points – thus their extension to selecting multiple data points required an unmotivated heuristic approach. A complementary line of research revolves around feature based interpretation of models. Instead of focusing on choosing representative data points, the goal is to reveal which features are important for the prediction Ribeiro et al. [24]. Recently, Kim et al. [16] also made use of the unweighted MMD function to propose selection of prototypes and criticisms. While their approach can be used for exploratory analysis of the data, it has not been extended for explaining a model. Their focus, moreover, is on the use of criticisms in addition to examples as a vital component of exploring datasets.

Fisher kernels were proposed to exploit the implicit embedding of a structured object in a generative model for discriminative purposes [13], and have since been applied successfully in a variety of applications [23]. The goal is to design a kernel for generative models of structured objects that captures the "similarity" for the said objects in the corresponding embedding space. The kernel itself can then be used out of the box in discriminative models such as Support vector machines.

2 Background

In this section, we provide an overview of the technical background required for our setup. We begin by fixing some notation. We represent sets using sans script fonts *e.g.* A, B. Vectors are represented using lower case bold letters *e.g.* $\mathbf{x}, \mathbf{y}$, and matrices are represented using upper case bold letters *e.g.* $\mathbf{X}, \mathbf{Y}$. Non-bold face letters are used for scalars *e.g.* j, M, r and function names *e.g.* $f(\cdot)$.

2.1 Fisher Kernels

The notion of similarity that Fisher kernels employ is that if two objects are *structurally* similar, then slight perturbations in the neighborhood of the fitted parameters $\hat{\theta} := \arg\max \log p(\mathbf{X}|\theta)$, would impact the fit of the two objects similarly. In other words, the feature embedding $\mathbf{f}_i := \frac{\partial \log p(\mathbf{X}_i|\theta)}{\partial \theta}|_{\theta=\hat{\theta}}$, for an object $\mathbf{X}_i \rightarrow \mathbf{f}_i$ can be interpreted as a *feature mapping* which can then be used to define a similarity kernel by a weighted dot product:

$$\kappa(\mathbf{X}_i, \mathbf{X}_j) := \mathbf{f}_i^\top \mathcal{I}^{-1} \mathbf{f}_j,$$

where the matrix $\mathcal{I} := \mathbb{E}_{p(\mathbf{X})}[\frac{\partial \log p(\mathbf{X}|\theta)}{\partial \theta}^\top \frac{\partial \log p(\mathbf{X}|\theta)}{\partial \theta}]$ is the Fisher information matrix. The information matrix serves to re-scale the dot product, and is often taken as identity as it loses significance in limit [13]. The corresponding kernel is then called the *practical* Fisher kernel and is often used in practice. We note, however, that dropping $\mathcal{I}$ had significant impact on performance in our method, so we employ the full kernel. However, the practical Fisher Kernel is important to mention here. As we show in Section 5, using the practical Fisher Kernel recovers the influence function based approach to interpretability [17] as a special case. Another interpretation of the Fisher kernel is that it defines the inner product of the directions of gradient ascent over the Riemannian manifold that the generative model lies in [25].

While appropriate feature mapping is crucial for predictive tasks, we observe that it is also is vital for interpretability. Fisher kernels are ideal for our task because they seamlessly extract model-induced data similarity from trained model that we wish to interpret. To further motivate that such a task can not be trivially performed by a something like a parameter sweep over RBF kernels i.e. without supervision, we perform a simple toy experiment illustrated in Figure 1.

2.2 Bayesian Quadrature

Bayesian quadrature [22] is a method used to approximate the expectation of a function by a weighted sum of a few evaluations of the said function. Say a function $f : \mathcal{X} \rightarrow \mathbb{R}$ is defined on a measurable space $\mathcal{X} \subset \mathbb{R}^d$. Consider the integral:

$$\mathbb{E}[f(\mathbf{x})] = \int_{\mathcal{X}} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \approx \sum_{i=1}^{n} w_i f(\mathbf{x}_i), \qquad (1)$$

where w_i are the weights associated with function evaluations at $\mathbf{x}_i$. Using $w_i = 1/n$ and randomly sampling $\mathbf{x}_i$ recovers the standard Monte Carlo integration. Other methods include kernel herding [3] and quasi-Monte carlo [7], both of which use $w_i = 1/n$ but use specific schemes to draw $\mathbf{x}_i$. Bayesian quadrature allows one to consider a non-uniform w_i given a functional prior for $f(\cdot)$. The samples $\mathbf{x}_i$ can then be chosen as the ones that minimize the posterior variance [12] as we shall see in the sequel. The corresponding weights can be calculated directly from the posterior mean. We impose a Gaussian Process prior on the function as $f \sim \text{GP}(0, k)$ with a kernel function $k(\cdot, \cdot)$. The algorithm SBQ proceeds as follows. Say we have already chosen n points: $\mathbf{x}_i, i \in [n]$. The posterior of f given the evaluations $f(\mathbf{x}_i)$ has the mean function:

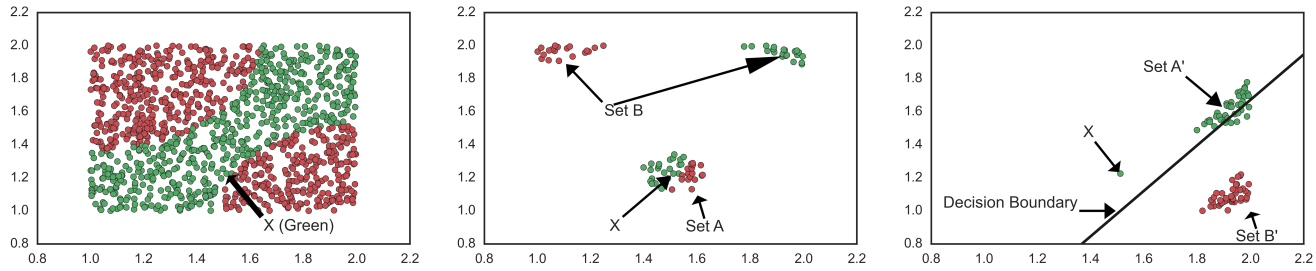


Figure 1: A toy experiment to illustrate the usefulness of Fisher space mapping. [Left] 1200 samples on U[1,2] $\times$ U[1,2] with two labels - Green and Red as illustrated. A specific green point X is selected for further experiment. [Mid] Closest 40 (Set A) and farthest 40 points (Set B) in terms of RBF kernel similarity. A distance based kernel such as RBF would yield these points as most and least similar to X respectively. [Right] Closest 40 (Set A') and farthest 40 (Set B') to X in terms of the Fisher kernel similarity computed from a fitted logistic regression model. The decision boundary for the logistic regression is also presented. It predicts everything below it as red, and everything above it as green. The Fisher "closeness" here takes into account the label of the points as well as the log-likelihood gradient on the contour of the loss function and its direction for each point. Note that for points exactly on the boundary, their gradient and Fisher similarity with all other points will be 0.

$$\hat{f}(\mathbf{x}) = \mathbf{k}^\top \mathbf{K}^{-1} \mathbf{f},$$

where $\mathbf{f}$ is the vector of function evaluations $f(\mathbf{x}_i)$, $\mathbf{k}$ is the vector of kernel evaluations $k(\mathbf{x}, \mathbf{x}_i)$, and $\mathbf{K}$ is the kernel matrix with $\mathbf{K}_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$.

We now focus on sampling the points $\mathbf{x}_i$. The quadrature estimate provides not only the mean, but the full distribution as its posterior. The posterior variance can be written as:

$$\text{cov}(\mathbf{x}, \mathbf{y}) = k(\mathbf{x}, \mathbf{y}) - k(\mathbf{x}, \mathbf{X}) \mathbf{K}^{-1} k(\mathbf{X}, \mathbf{y}),$$

where $\mathbf{X}$ is the matrix formed by stacking $\mathbf{x}_i$, and the kernel function notation is overloaded so that $k(\mathbf{X}, \mathbf{y})$ represents the column vector obtained by stacking $k(\mathbf{x}_i, \mathbf{y})$. The posterior over the function f also yields a posterior over the expectation over f defined in (1). For convenience, define the set $\mathsf{S}_j := \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_j\}$. Say $Z(\mathsf{S}_j) := \sum_j w_j f(\mathbf{x}_j)$. Then, it is straightforward to see $\mathbb{E}[Z(\mathsf{S}_n)] = \mathbf{z}^\top \mathbf{K}^{-1} \mathbf{f}$, where $\mathbf{z}_i := \int k(\mathbf{x}, \mathbf{x}_i) p(\mathbf{x}) d\mathbf{x}$. Note that the weights in (1) can be written as $w_i = \sum_j \mathbf{z}_j [\mathbf{K}^{-1}]_{ij}$.

We can write the variance of $Z(\mathsf{S}_n)$ as:

$$\text{var}(Z(\mathsf{S}_n)) = \iint k(\mathbf{x}, \mathbf{y}) p(\mathbf{x}) p(\mathbf{y}) d\mathbf{x} d\mathbf{y} - \mathbf{z}^\top \mathbf{K}^{-1} \mathbf{z}. \tag{2}$$

The algorithm Sequential Bayesian Quadrature (SBQ) samples for the points $\mathbf{x}_i$ in a greedy fashion with the goal of minimizing the posterior variance of the computed approximate integral:

$$\mathbf{x}_{n+1} \leftarrow \arg\min_{\mathbf{x} \in \mathcal{X}} \text{var}(Z(\mathsf{S}_n \cup \{\mathbf{x}\})).$$

3 Prototype Selection using Fisher Kernels

In this section, we present our method to select sample representatives using Fisher kernels. For a loss function $\ell(\theta, \mathbf{x})$, where θ are the parameters of the model and $\mathbf{x}$ is the data, to train a parametric model one would minimize the expected loss:

$$\min \mathbb{E}_{p(\mathbf{x})} \ell(\theta, \mathbf{x}), \tag{3}$$

where $p(\mathbf{x})$ is the data distribution. Since we usually do not have access to the true data distribution, $p(\mathbf{x})$ is typically the empirical data distribution $p(\mathbf{x}) = \frac{1}{n}\delta(\mathbf{x})$, where $\delta(\cdot)$ is 1 if $\mathbf{x}$ exists in the dataset, and 0 otherwise, and n is the size of the dataset. Our goal in this work is to approximate the integral (3) over the test or validation set (which specifies the distribution p for us) using a weighted sum of a *few* points from the *training* dataset (1). Note that while the training samples in general have measure 0 in the test or the validation set distribution in the euclidean space, the smoothening GP prior over the embedding space still allows for samples to be generated from the former to approximate the latter.

For the kernel function in the GP prior in Bayesian Quadrature, we use the Fisher kernel of the trained parametric model. SBQ selection strategy inherently establishes a trade off between selecting data points that are representative of the parametric fit and diversity of the selected points. To see this, consider the SBQ cost function (2). At every new selection $\mathbf{x}_{j+1}$, one one hand, the cost function rewards the selection of data points which are clustered closer together in the feature mapping space to increase the value of $\mathbf{z}$ which in turn decreases variance. However, on the other hand, selecting points close to each other decreases the eigenvalues of $\mathbf{K}^{-1}$ thereby increasing variance [12].

Thus, the SBQ seeks a tradeoff between these terms.

3.1 An Efficient Greedy Algorithm

In this section, we provide a practical greedy algorithm to select representative prototypes using SBQ to optimize (2). Note that the first term is constant w.r.t to S_n. Moreover, recall that the target distribution for us is$p(\mathbf{x}) = \frac{1}{n}\delta(\mathbf{x})$, where n is the size of the test/validation set. Thus, we can re-write $\mathbf{z}_i = \frac{1}{n}\sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j)$ for each i in training and each j in the test set. This can be pre-computed by a row or column sum over the kernel of the entire dataset in $O(nt)$ time and stored as vector of size t to speed up later computation, where t is the size of the training set. Our greedy cost function at step $j+1$ is thus:

$$i^{\star}_{j+1} \leftarrow \arg\max_{\substack{i\in[t]\setminus\mathsf{S}_j \\ \mathsf{S}=\mathsf{S}_j\cup i}} \mathbf{z}_{\mathsf{S}}^{\top}[\mathbf{K}_{\mathsf{SS}}^{-1}]\mathbf{z}_{\mathsf{S}}. \tag{4}$$

The solution set is then updated as $\mathsf{S}_{j+1} = \mathsf{S}_j \cup \{i^{\star}_{j+1}\}$. The optimization (4) requires an inverse of the kernel matrix of already selected data points which can be computationally expensive. However, we can use the following result from linear algebra about block matrix inverses to speed up operations.

Proposition 1. *For an invertible matrix* $\mathbf{A}$*, a column vector* $\mathbf{b}$*, and a scalar* c*, let* $d = c - \mathbf{b}^{\top}\mathbf{A}^{-1}\mathbf{b}$*, then*

$$\begin{bmatrix}\mathbf{A} & \mathbf{b}\\ \mathbf{b}^{\top} & c\end{bmatrix}^{-1} = \frac{1}{d}\begin{bmatrix} d\mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{b}\mathbf{b}^{\top}\mathbf{A}^{-1} & \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{b}^{\top}\mathbf{A}^{-1} & 1\end{bmatrix}$$

Proposition 1 allows us to build the inverse of the kernel $\mathbf{K}$ in (4) greedily. The full algorithm is presented in Algorithm 1.

Algorithm 1 obviates the need for taking explicit inverses and only requires an oracle access to the kernel function. The algorithm itself is inherently embarrassingly parallelizable over multiple cores. We study guarantees for the algorithm in Section 4 which also motivates its more scalable variants.

Algorithm 1 Greedy Prototype Selection

INPUT: Data $\{\mathbf{x}_i\}$, kernel function $k(\cdot,\cdot)$, number of selections to make k, t is the size of the training set
//Pre-compute $\mathbf{z}_i$
$\mathbf{z}_i = \frac{1}{n}\sum_j k(\mathbf{x}_i, \mathbf{x}_j) \forall i \in$ training and $j \in$ test
// Build solution set S greedily. Maintain current inverse(K) at each iteration as inv$\mathbf{K}$
$\mathsf{S} = \emptyset$, inv$\mathbf{K} = []$
for $i = 1 \dots k$ **do**
  $j^{\star} = -1$, MAX $= -\infty$
  for $j \in [t]\setminus\mathsf{S}$ **do**
    $\mathbf{t} = \mathbf{z}[\mathsf{S}\cup j]$
    $\mathbf{b} = k(\mathbf{X}_{\mathsf{S}}, \mathbf{x}_j)$, $c = k(\mathbf{x}_j, \mathbf{x}_j)$, $\mathbf{A}^{-1} =$ inv$\mathbf{K}$ Get $\mathbf{T}$ as the updated inverse using Prop. 1
    If $(\mathbf{t}^{\top}\mathbf{T}\mathbf{t} >$ MAX$)$, $j^{\star} = j$, MAX $= \mathbf{t}^{\top}\mathbf{T}\mathbf{t}$
  end for
  Write $\mathbf{b} = k(\mathbf{X}_{\mathsf{S}}, \mathbf{x}_{j^{\star}})$, $c = k(\mathbf{x}_{j^{\star}}, \mathbf{x}_{j^{\star}})$, $\mathbf{A}^{-1} =$ inv$\mathbf{K}$
  Update: inv$\mathbf{K}$ using Prop. 1, $\mathsf{S} = \mathsf{S}\cup j^{\star}$
end for
return S

4 Analysis

The greedy algorithm described in Algorithm 1 while being simple also has interesting optimization guarantees that make it attractive to use in practice. In this section, we provide convergence guarantees for the cost function (2) as n increases. Typically for functions like these in the general case, the candidate set of atoms used to build the approximation is uncountably infinite - any possible sample from the underlying density is a candidate. As such, the convergence results are based on using Frank-Wolfe analysis on the marginal polytope [2]. However, for us the underlying set of candidate atoms are discrete points, which are at worst countably infinite. As such, for this special case, it is worth analyzing if we can provide better rates than the general available guarantees. It turns out that this is indeed possible. We are able to leverage recent research in discrete optimization to indeed provide a linear convergence rate for the forward greedy algorithm.

Recall our set optimization function (from (4)) is:

$$g(\mathsf{S}) := \max_{\substack{\mathsf{S}\subset[t]\\ |\mathsf{S}|\le r}} \mathbf{z}_{\mathsf{S}}[\mathbf{K}_{\mathsf{SS}}^{-1}]\mathbf{z}_{\mathsf{S}}, \tag{5}$$

where n is the set of candidate training data points. We write $\mu_p := \iint k(\mathbf{x},\mathbf{y})p(\mathbf{x})p(\mathbf{y})d\mathbf{x}d\mathbf{y}$. For the RKHS induced by the kernel $\mathcal{H}$, we can equivalently re-write the cost function as [12, 2]:

$$\min_{\substack{\mathsf{S}\subset[n]\\ |\mathsf{S}|\le r}} v(\mathsf{S}) := \mu_p - \sum_{i\in\mathsf{S}} w_i\mathbf{z}_i \tag{6}$$

For a matrix $\mathbf{A}$, the smallest (largest) k-sparse eigenvalues is min (max) of $\frac{\mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\mathbf{x}^{\top}\mathbf{x}}$ under the constraints $\|\mathbf{x}\|_0 \le k$, and $\mathbf{x} \neq 0$. Note that we can write $v(\emptyset) = \mu_p$. We present our convergence guarantee next.

Theorem 2. *Say* $\mathcal{H}$ *is finite dimensional and has bounded norm i.e.* $\forall \nu \in \mathcal{H}$, $\|\nu\|_{\mathcal{H}} < \infty$. *Let* m *be the smallest* $2r$ *sparse eigenvalue and* M *be the largest* $r+1$*-sparse eigenvalues of the kernel matrix* $\mathbf{K}$ *of the training set. If* S_G *of size* k *is the set returned by Algorithm 1 and* $\mathsf{S}^{\star}$ *of size* r *is the optimal solution of* (6)*, then if* $k \ge \frac{M}{m} r \log\frac{1}{\epsilon}$, $v(\mathsf{S}_G) - v(\mathsf{S}^{\star}) \le \epsilon(v(\emptyset) - v(\mathsf{S}^{\star}))$.

Discussion: Theorem 2 provides exponential convergence for the cost function $v(\cdot)$. For the same objective, using Frank-Wolfe on the marginal polytope, the best known guarantees in the most general case are $O(1/T)$ for finite dimensional bounded Hilbert spaces [2]. In the special case when the optimum lies in the relative interior, we do get faster exponential convergence. Theorem 2 provides an alternative condition that is sufficient for exponential convergence for the case when the optimum μ_p lies at the boundary of the marginal polytope instead of in its relative interior i.e. it is linear combination of r atoms. The lower sparse eigenvalue condition is a union bound, and only requires to hold over the greedy selection set plus any r sized subset.

4.1 Scalability

For massively large real world datasets, the standard greedy algorithm (SBQ) may be prohibitively slow. In addition to run time, there are also memory considerations. SBQ requires building and storing an $O(t^2)$ sized kernel matrix over the training set of size t. We can use alternative variants of the greedy algorithm that are either faster with some compromise on the convergence rate or can distribute the kernel over multiple machines. These variants are presented in Table 1 with their corresponding references. To the best of our knowledge, these variants have not been suggested for solving the problem (1) before and may be of independent interest. The convergence rates are obtained similar to the proof of Theorem 2 by plugging in respective approximation guarantees in lieu of Lemma 7 in the appendix.

5 Relationship with Influence functions

Influence functions [5] have recently been proposed as a tool for interpreting model predictions [17]. Since our goal is also the same, it is interesting to ask if there is a relationship between the two approaches. For selecting the most influential training point for a given test point, influence functions approximate infinitesimal upweighting of which training point has the most effect on prediction of the test point in question. In this section, we show that our method recovers this influence function approach used by Koh & Liang [17] for selecting influential training data points. In addition, we also show how adversarial training side attacks proposed by Koh & Liang [17] by perturbing features of training data points can be re-interpreted as a standard adversarial attack in the RKHS induced by the Fisher Kernel. Our analysis yields new insights about the influence function based approach and also establishes the importance of the Fisher space for robust learning.

5.1 Choosing training data points

We briefly introduce the influence function approach for model interpretation. For simplicity, we re-use the notation suggested by Koh & Liang [17]. Let $\mathbf{z}_{\text{test}}$ be the test data point in question, $\mathsf{S}_{\text{train}}$ be the training set, $L(\mathbf{z}, \theta)$ be the loss function fitted on the training set, $\hat{\theta}$ be the optimizer of $L(\mathsf{S}_{\text{train}}, \theta)$, $\mathbf{H}_\theta$ be the Hessian of the loss function evaluated at θ, then the most influential training data point is the solution of the optimization problem:

$$\max_{\mathbf{z} \in \mathsf{S}_{\text{train}}} \nabla_\theta L(\mathbf{z}, \hat{\theta}) \mathbf{H}_{\hat{\theta}}^{-1} \nabla_\theta L(\mathbf{z}_{\text{test}}, \hat{\theta}) \tag{7}$$

We compare the two discrete optimization problems (5) and (7). Even though (5) uses first order information only while (7) uses both first order and second order information about the loss function, the following proposition illustrates a connection.

Proposition 3. *If the loss function $L(\cdot)$ takes the form of a negative log-likelihood function, then $[H_{\hat{\theta}}]_{ij} = \nabla_{\theta_i} L(\mathsf{S}_{\text{train}}, \hat{\theta})^\top \nabla_{\theta_j} L(\mathsf{S}_{\text{train}}, \hat{\theta})$, where we have overloaded the notation $L(\mathsf{S}_{\text{train}}, \theta) = \frac{1}{|\mathsf{S}_{\text{train}}|} \sum_t L(\mathbf{z}_t, \theta)$.*

Proof. Let $L(\mathbf{z}, \theta) := -\log p(\mathbf{z}, \theta)$, since it takes form of a negative LL function. Then, since $\hat{\theta}$ is the optimizer of $L(\mathsf{S}_{\text{train}}, \theta)$,

$$\begin{aligned}
&\nabla_{\theta_i} L(\mathsf{S}_{\text{train}}, \hat{\theta}) = 0 \\
\implies &\nabla_{\theta_i} \sum_t -\log p(\mathbf{z}_t, \hat{\theta}) = 0 \\
\implies &\nabla_{\theta_j} \nabla_{\theta_i} \sum_t -\log p(\mathbf{z}_t, \hat{\theta}) = 0 \\
\implies &\nabla_{\theta_j} \sum_t \frac{-1}{p(\mathbf{z}_t, \hat{\theta})} \nabla_{\theta_i} p(\mathbf{z}_t, \hat{\theta}) = 0 \\
\implies &\sum_t \frac{\nabla_{\theta_i} \nabla_{\theta_j} p(\mathbf{z}_t, \hat{\theta})}{p(\mathbf{z}_t, \hat{\theta})} = \sum_t \frac{\nabla_{\theta_i} p(\mathbf{z}_t, \hat{\theta})^\top \nabla_{\theta_j} p(\mathbf{z}_t, \hat{\theta})}{p(\mathbf{z}_t, \hat{\theta})^2},
\end{aligned}$$

from which the result directly follows. □

From Proposition 1, it is easy to see that the optimization problems (5) and (7) are the same under some conditions. To be more precise, we can make the following statement. If the cost function $L(\cdot, \cdot)$ is in the form of a negative log-likelihood function, (7) is a special case of (5) with the practical Fisher kernel (see Section 2.1) when the test set is of size 1, and $r = 1$.

This equivalence gives several insights about influence functions that were not known before: (1) it generalizes influence functions to multiple data points for both test and training sets in a principled way and provides a probabilistic foundation to the method, (2) it establishes the importance of the induced RKHS by the Fisher kernel by re-interpreting the influence function optimization problem as $\min_{\mathbf{z} \in \mathsf{S}_{\text{train}}} \| \mathbf{z}_{\text{test}} - \mathbf{z} \|_{\mathcal{H}}$ (see Lemma 4 in the appendix), (3) for negative LL functions, it renders the expensive calculation of the Hessian in the work by Koh & Liang [17]

Algorithm	Runtime	Memory required	Convergence rate
SBQ (Algorithm 1)	$O(k^3t)$	$O(t^2+n)$	$O(\lambda log 1/\epsilon)$
Matching Pursuit [8]	$O(k^2t)$	$O(t^2+n)$	$O(\lambda log 1/\epsilon)$
δ-Stochastic Selection [14]	$O(kt \log 1/\delta)$	$O(t^2+n)$	$O(\lambda log 1/(\delta\epsilon))$
Distributed (l machines) [14]	$O(\frac{kt}{l})$	$O(\frac{t^2}{l^2}+n)$	$O(\lambda log \frac{1}{\epsilon})$

Table 1: Greedy variants for prototype selection. $\lambda = \frac{M}{m}$, n is the test set size, t is the size of the training set. Convergence rate refers to number of iterations needed to get ϵ accuracy. For Stochastic and Distributed variants, the guarantee is in expectation.

redundant since by Proposition 3, first order information suffices, (4) it provides theoretical approximation guarantees (see Lemma 7 in the appendix) for selection of multiple training data points, in constrast to Koh & Liang [17] who made multiple selections greedily only as a heuristic.

5.2 Unified view of adversarial attacks

Given a test data point $\mathbf{z}$, an adversarial example is generated by adding a small perturbation as $\tilde{\mathbf{z}} = \mathbf{z} + \epsilon_z$, where ϵ_z is a small perturbation of $\mathbf{z}$ so that for $\tilde{\mathbf{z}}$ is indistinguishable from $\mathbf{z}$ by a human, but causes the model to make an incorrect prediction on $\mathbf{z}$ [9]. For training data attacks, $\mathbf{z}$ is a training data point that is perturbed to make an incorrect prediction on a test data point. For a loss function $\ell(\mathbf{z})$, a test side attack for perturbing a test data point $\mathbf{z}_{\text{test}}$ would solve the optimization problem:

$$\max_{\|\mathbf{z}-\mathbf{z}_{\text{test}}\|_\infty \leq \epsilon} \ell(\mathbf{z}) \tag{8}$$

While the optimization (8) is hard in general, typically a few iterations of projected gradient ascent or FGSM are applied. We refer to the recent work by Madry et al. [19] for details.

For training side attacks, Koh & Liang [17] perform the following iterative update:

$$\tilde{\mathbf{z}} \leftarrow \Pi(\tilde{\mathbf{z}} + \alpha \text{sign}(\mathcal{L}(\tilde{\mathbf{z}}, \mathbf{z}_{\text{test}}))), \tag{9}$$

where $\mathbf{z} = (x, y)$ is a candidate training example to perturb in x, $\mathbf{z}_{\text{test}}$ is the target test example, Π is the projection operator onto the set of valid images, α is a fixed step size, and $\mathcal{L}(\tilde{\mathbf{z}}, \mathbf{z}_{\text{test}}) := \nabla_\theta L(\mathbf{z}, \hat{\theta}) \mathbf{H}_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(\mathbf{z}_{\text{test}}, \hat{\theta})$.

Using the results in Section 5.1, it is straightforward to see that the if we use $\ell(\mathbf{z}) = \| \mathbf{z}_{\text{test}} - \mathbf{z} \|_{\mathcal{H}}$, where $\mathcal{H}$ is the RKHS induced by the practical Fisher kernel, and change the constraint as a perturbation over a training example instead of the test example, we recover the iterative step (9) as a special case of projected gradient ascent steps to solve (8).

This equivalence provides a unified view of both training and test side attacks. As such, the large literature on robust learning against test side attacks can be applied to robustness against training side attacks as well. Moreover our framework also provides a principled way to do training side attacks to target multiple test set examples, instead of attacking individual test points separately.

6 Experiments

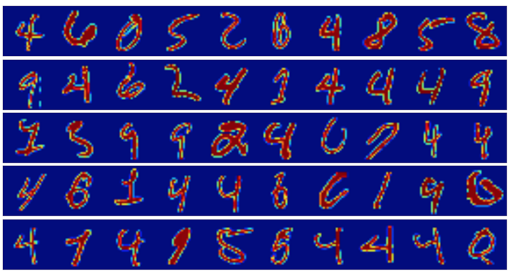
We present empirical use cases of our framework. We chose the experiments to illustrate the flexibility of our framework, as well as to emphasize its generalization capacity over and above influence functions. As such, we present experiments that make use of set influence (as opposed to single data point influence) for data cleaning and summarization (Sections 6.1,6.3). To illustrate potential benefit of using the full Fisher kernel as opposed to the simplified practical Fisher kernel as used by the influence functions, we present evaluation for a use case for fixing mislabelled examples as presented by Koh & Liang [17] (Section 6.2).

6.1 Data Cleaning: removing malicious training data points

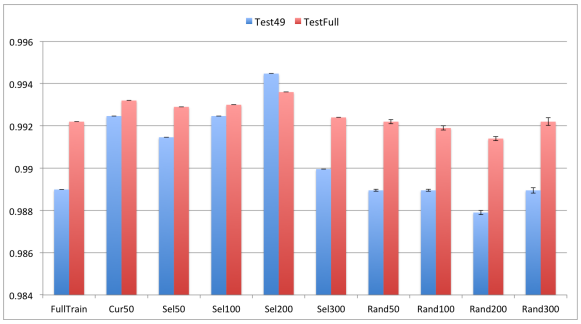
In this section, we present experiments on the MNIST dataset to illustrate the effectiveness of our method in interpreting model behavior for the test population. Some of the handwritten digits in MNIST are hard even for a human to classify correctly. Such points can adversely affect the training of the classifier, leading to lower predictive accuracy. Our goal in this experiment is to try to identify some such misleading training data points, and remove them to see if it improves predictive accuracy. To illustrate the flexibility of our approach, we focus only on the digits 4 and 9 in the test data which were misclassified by our model, and then select the training data points responsible for those misclassifications.

The MNIST data set [18] consists of images of handwritten digits and their respective labels. Each image is a 28×28 pixel array. There are 70000 images in total, split into 60000 training examples and 10000 test examples. The 10 digits are about evenly represented in both the training and the test data.

For the classification task, we use tensorflow [1] to build a 2 layer convolutional network with 2×2 max pooling followed by a fully connected layer and the softmax layer.



(a) A subset of selected prototypes responsible for misclassifying 4s and 9s in the test set



(b) Accuracy fractions on test data 4s and 9s (Test49), and the full test set after removing random (Rand), algorithm selected (Sel), or Curated (Cur) prototypes.

Figure 2: MNIST experiment for selecting malicious training data points.

The convolutions use a stride of 1 followed by padding of zeros to match the input size. We use dropout to avoid overfitting. The network was trained using the built-in Adam Optimizer for 20000 steps of batch size 100 each. For the entire test set, we obtain an accuracy of 0.9922, while for the subset of the test set consisting only of the chosen two digits 4 and 9, the accuracy is 0.9889.

After the training is completed, we obtain the gradients of the training and test data points w.r.t the parameters of the network by passing each point through the trained (and subsequently frozen) network. The obtained gradient vectors are used to calculate the Fisher kernel as detailed in Section 2.1. We then employ Algorithm 1 using the newly built Fisher kernel matrix between training and test datasets to obtain the top 300 *prototypes* i.e. data points from the training set that our algorithm deems most responsible for misclassifying 4s and 9s.

To check if these points are indeed misguiding the model, we remove the top $50, 100, 200, 300$ of the selected points from the training data and retrain the model to retest on the test set. These numbers are reported as Sel50, Sel100, Sel200, Sel300 in Figure 2b. Indeed we see an improvement in the test accuracy till Sel200 indicating the importance of removing the selected potential malicious points from the training set, and a subsequent decay in performance for Sel300 most likely due to removal of too many useful points

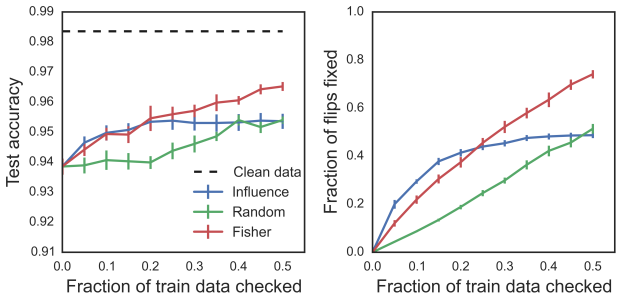


Figure 3: Comparison of SBQ compared to Influence functions on the task of fixing flipped labels.

in addition to malicious ones. To compare, we also remove the respective number of points randomly and repeat the experiment. Removal of random points from the training data led to a general decay in the predictive accuracy.

Finally, we manually selected 50 points from the chosen 300 points as the curated set based on how ill-formed the digits were (see Figure 2a). Removing these points from the training set before re-training and testing gives predictive accuracy is reported as Cur50 comparable to Sel100, but still worse than Sel200, indicating that the algorithm identified more malicious points in top-200 selected than our manually chosen 50 points.

6.2 Fixing Mislabeled Examples

In this experiment, we use our framework to detect and fix mislabeled examples. Labor intenstive labeling tasks naturally result in mislabels, especially in real-world datasets. These data points may cause poor performance and degradation of the model. We show that our method can be successfully used for this purpose, showing improvement over the recent results by Koh & Liang [17].

We use a small correctly labeled validation set to identify examples from the large training set that are likely mislabeled. We first train a classifier on the noisy training set, and predict on the validation set. We then employ Algorithm 1 to identify training examples that were responsible for making incorrect predictions on the validation set. The potentially mislabeled data points are then chosen by the output of our method. Curation is then simulated on the selected examples in order of selections made (similar to the approach by Koh & Liang [17]), and if the label was indeed wrong, it is fixed. We report on the number of training data points selected vs fixed (the precision metric for incorrectly labeled points) and the respective improvement in unseen test data accuracy.

For evaluation, we use `enron1` email spam dataset used by Koh & Liang [17] and compare our results to their reported results. The dataset consists of 4137 training points and 1035 test points. We randomly select 500 data points

from the training set as the clean *curated* data. From the remaining training data points, we randomly flip the labels of 20% of the data. We then use our method and the baselines to select several candidates for curation. We report the number of fixes made after these selections and the corresponding test predictive accuracy. The baselines are selection by (i) top self influence measures [17], and (ii) random selection of datapoints. The curation data is used as part of the training by all the methods. No method had access to the test data. As showing in Figure 3, our algorithm consistently performs better in test accuracy and the fraction of flips fixed as more and more data is curated.

6.3 Data Summarization

In this section, we perform the task of training data summarization. Our goal is to select a few data samples that represent the data distribution *sufficiently* well, so that a model built on the selected subsample of the training data does not degrade too much in performance on the unseen test data. This task is complimentary to the task of interpretation, wherein one is interested in selecting training samples that explain some particular predictions on the test set. Since we are interested in approximating the test distribution using a few samples from a training set with the goal of predictive accuracy under a given model, our framework of Sequential Bayesian Quadrature using Fisher kernels is directly applicable.

Another method that also aims to do training data summarization is that of coreset selection [11], albeit with a different goal of reducing the training data size for optimization speedup while still maintaining guaranteed approximation to the training likelihood. Since the goal itself is optimization speedup, coreset selection algorithms typically employ fast methods while still trying to capture the data distribution by proxy of the training likelihood. Moreover, the coreset selection algorithm is usually closely tied with the respective model as opposed to being a model-agnostic method like ours.

To illustrate that coreset selection falls short on the goal of competitively estimating the data distribution, we employ our framework to the problem of training data summarization under logistic regression, as considered by Huggins et al. [11] using coreset construction. We experiment using two datasets `ChemReact` and `CovType`. `ChemReact` consists of 26733 chemicals each of feature size 100. Out of these, 2500 are test data points. The prediction variable is $0/1$ and signifies if a chemical is reactive. `CovType` has 581012 examples each of feature size 54. Out of these, 29000 are test points. The task is to predict whether a type of tree is present in each location or not.

In each of the datasets, we further randomly split the training data into 10% validation and 90% training. For the larger `CovType` data, we note that selecting about 20,000 training points out of the training set achieves about the same performance as the full set. Hence, we work with randomly selected 20,000 points for speedup. We train the logistic regression model on the new training data, and use the validation set as a proxy to the unseen test set. We build the kernel matrix $\mathbf{K}$ and the affinity vector $\mathbf{z}$, and run Algorithm 1 for various values of k. For the baselines, we use the coreset selection algorithm and random data selection as implemented by Huggins et al. [11]. The results are presented in Figure 4. We note that our algorithm yields a significantly better predictive performance compared to random subsets and coresets [11] with the same size of the training subset across different subset sizes.

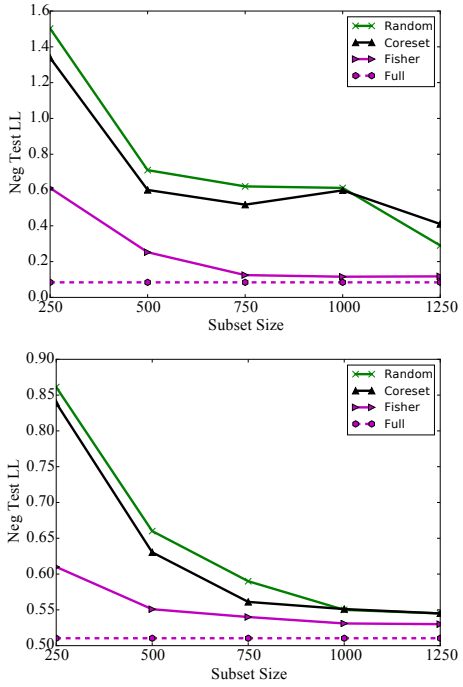


Figure 4: Performance for logistic regression over two datasets (top is `ChemReact` while bot is `CovType`) of our method (Fisher) vs coreset selection [11] and random data selection. 'Full' reports the numbers for training with the entire training set. Fisher (proposed) achieves much better test LL performance across different subset sizes.

Conclusion: We proposed a novel principled approach for examining sets of training examples that influence an entire test set given a trained black-box model – extending a notable recently proposed per-example influence to set-wise influence. We also presented novel convergence guarantees for SBQ and more scalable algorithm variants. Empirical results were presented to highlight the utility of the proposed approach for black-box model interpretability and related tasks. For future work, we plan to investigate the use of model criticisms to provide additional insights into the trained models.

References

[1] Abadi, Martin, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G., Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. Tensorflow: A system for large-scale machine learning. 2016.

[2] Bach, Francis, Lacoste-Julien, Simon, and Obozinski, Guillaume. On the equivalence between herding and conditional gradient algorithms. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pp. 1355–1362, 2012.

[3] Chen, Yutian, Welling, Max, and Smola, Alexander J. Super-samples from kernel herding. In *UAI*, 2010.

[4] Cohen, M.S., Freeman, J.T., and Wolf, S. Metarecognition in time-stressed decision making: Recognizing, critiquing, and correcting. *Human Factors*, 1996.

[5] Cook, R. Dennis and Weisberg, Sanford. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4): 495–508, 1980.

[6] Das, Abhimanyu and Kempe, David. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. In *ICML*, February 2011.

[7] Dick, Josef and Pillichshammer, Friedrich. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, New York, NY, USA, 2010.

[8] Elenberg, Ethan R., Khanna, Rajiv, Dimakis, Alexandros G., and Negahban, Sahand. Restricted Strong Convexity Implies Weak Submodularity. *Annals of Statistics*, 2018.

[9] Goodfellow, Ian J., Shlens, Jonathon, and Szegedy, Christian. Explaining and harnessing adversarial examples. *ICLR*, 2015.

[10] Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., and Smola, A. A kernel method for the two-sample problem. *JMLR*, 2008.

[11] Huggins, Jonathan H., Campbell, Trevor, and Broderick, Tamara. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4080–4088, 2016.

[12] Huszar, Ferenc and Duvenaud, David K. Optimally-weighted herding is bayesian quadrature. In *UAI*, 2012.

[13] Jaakkola, Tommi S. and Haussler, David. Exploiting generative models in discriminative classifiers. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pp. 487–493, Cambridge, MA, USA, 1999. MIT Press.

[14] Khanna, Rajiv, Elenberg, Ethan R., Dimakis, Alexandros G., Neghaban, Sahand, and Ghosh, Joydeep. Scalable Greedy Support Selection via Weak Submodularity. *AISTATS*, 2017.

[15] Kim, B., Rudin, C., and Shah, J.A. The Bayesian Case Model: A generative approach for case-based reasoning and prototype classification. In *NIPS*, 2014.

[16] Kim, Been, Khanna, Rajiv, and Koyejo, Oluwasanmi O. Examples are not enough, learn to criticize! criticism for interpretability. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2280–2288. 2016.

[17] Koh, Pang Wei and Liang, Percy. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1885–1894, 2017.

[18] LeCun, Yann, Bottou, LÃľon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pp. 2278–2324, 1998.

[19] Madry, Aleksander, Makelov, Aleksandar, Schmidt, Ludwig, Tsipras, Dimitris, and Vladu, Adrian. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[20] Nemhauser, George L, Wolsey, Laurence A, and Fisher, Marshall L. An analysis of approximations for maximizing submodular set functionsâĂŤi. *Mathematical Programming*, 14(1):265–294, 1978.

[21] Newell, A. and Simon, H.A. *Human problem solving*. Prentice-Hall Englewood Cliffs, 1972.

[22] O'Hagan, A. Bayes-hermite quadrature. 29, 11 1991.

[23] Perronnin, Florent, Sánchez, Jorge, and Mensink, Thomas. Improving the fisher kernel for large-scale image classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, pp. 143–156, Berlin, Heidelberg, 2010.

[24] Ribeiro, Marco Tulio, Singh, Sameer, and Guestrin, Carlos. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1135–1144, 2016. ISBN 978-1-4503-4232-2.

[25] Shawe-Taylor, John and Cristianini, Nello. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.